Guaranteed invocation/consumption of nested, composite software services

DESCRIPTION

CROSS-REFERENCES TO RELATED APPLICATIONS

[Para 1] This application claims the benefit of U.S. Provisional Application Serial No. 60/481,349, filed on September 10, 2003, entitled "Guaranteed invocation of nested, composite software services", which is incorporated herein by reference.

BACKGROUND OF THE INVENTION

[Para 2] 1. Field of the Invention

[Para 3] The present invention relates to the field of software application development and, in particular, to service-oriented programming and guaranteed invocation/consumption of nested, composite software services.

[Para 4] 2. Description of the Related Art

[Para 5] The current art of asynchronous message-oriented communication provides for guaranteed messaging. Based on the current art, a messaging system such as IBM's MQ-Series will guarantee the once and only once processing of a message. With the advent of Web Services and standards-based, service-oriented communication, the same concept has been extended to the delivery and processing of the input data of a service upon consumption.

[Para 6] Service-oriented programming introduces the concept of nested composite software services, where a software service may contain other software services that may themselves be composite software services. The current art of guaranteed messaging and reliable service-oriented communication is based on guaranteed delivery of messages or inputs of a service and does not address guaranteed consumption/invocation of nested

composite services. In fact, addressing the guaranteed consumption/invocation of nested composite services is beyond the scope of the current approach to guaranteed messaging and reliable service-oriented communication.

[Para 7] Without innovative features such as guaranteed invocation/consumption of nested composite services, service-oriented programming, applications and architecture cannot address the needs of mission-critical applications.

SUMMARY OF THE INVENTION

[Para 8] A principle object of the present invention is to provide a method for guaranteeing the exactly once consumption/invocation of a software service, including, but not limited to, "Web services", where the software service may be a composite software service: a service that is composed of other services nested without a fixed limit on the depth of composition [FIGURE 3]. Furthermore, the present invention guarantees the exactly once consumption of the services contained within the composite service and possibly those services contained within the contained services, regardless of depth of containment, if the invocation of a containing service is declared to be guaranteed.

[Para 9] The present invention provides a persistent, nested context mechanism capable of remembering the state of execution corresponding to the invocation of nested composite services as it relates to guaranteed invocation.

[Para 10] Yet another object of the present invention is to provide semantic-based configurable behavior, through software service composition tools, for the automated guaranteed invocation of a software service within user specified limits.

[Para 11] Other objects and advantages of this invention will be set in part in the description and in the drawings that follow and, in part, will be obvious

from the description, or may be learned by practice of the invention. Accordingly, the drawings and description are to be regarded as illustrative in nature, and not as restrictive.

[Para 12] To achieve the foregoing objectives, and in accordance with the purpose of the invention as broadly described herein, the present invention provides methods, frameworks, and systems for providing a method for guaranteed invocation of optionally nested composite services. In preferred embodiments, this technique comprises: 1) associating the attribute required for the guaranteed invocation to the service interface definition; 2) providing the ability to overwrite those attributes for a service contained within another composite service, in the context of the containing service; 3) defining a wrapper interface with methods accommodating the guaranteed invocation of non-composite services; 4) creating a directed execution graph based on the definition of a composite service having one such graph per each embedded composite service; 5) creating a possibly nested Invocation Map corresponding to the state of execution of each instance of a composite service; 6) coupling a persistent context mechanism with each Invocation Map; 7) associating unique ids to each service invocation while breaking the invocation into steps based on the directed execution graph; 8) recording each step; 9) performing each step of invocation and marking the results through the persistent context mechanism where a result of the invocation of a contained service is recorded; and 10) upon a system failure, or inability to invoke a service due to a system failure such as connection failure, attempting to retry the invocation of the service from where it previously left the invocation, based on the state of the associated context mechanism; 11) upon the unknown state of the invocation of a particular service, invoking the service with the same id with which it was invoked prior to the unknown state.

[Para 13] The present invention will now be described with reference to the following drawings, in which like reference numbers denote the same element throughout. It is intended that any other advantages and objects of the present invention that become apparent or obvious from the detailed description or illustrations contained herein are within the scope of the present invention.

BRIEF DESCRIPTION OF THE DRAWINGS

[Para 14] FIGURE 1 depicts an example interface of a software service named 'GetStockQuote'.

[Para 15] FIGURE 2 shows visual composition of a software service using a service composition and assembly tool, HyperService™ Studio.

[Para 16] FIGURE 3 shows the nesting of other software services within a composite software service.

[Para 17] FIGURE 4 shows an example of attributes that can be associated with a software interface to declare guaranteed invocation behavior

DESCRIPTION OF THE PREFERRED EMBODIMENTS

[Para 18] The reliable and correct execution of composite software services, including, but not limited to Web services, is a key concern in the implementation of the emerging service-oriented systems that support building service-oriented software solutions. Addressing the reliable invocation/consumption of nested, composite software services at runtime requires an automation platform to guarantee the invocation of a composite software service. Furthermore, this guarantee must include at least once, as well as exactly once semantics for the invocation of each software service contained within a composite service regardless of the depth of containment. The current art of guaranteed messaging as it relates to message-oriented systems can only address the protocols required for the guaranteed invocation of an atomic software service, one that is not composed of other software services. The present invention goes beyond the current art and provides an automated, declarative method for guaranteeing the invocation of nested composite software services with once and exactly once semantics.

[Para 19] Assuming a visual software service composition and assembly tool, such as HyperService™ Studio, with a snapshot of a definition in FIGURE 2, and an automatic flow platform, such as the HyperService™ platform, available at www.nextaxiom.com, we will now describe how such a composition tool and platform can be extended to accommodate declarative, automated guaranteed invocation for optionally, nested composite software services.

[Para 20] Emerging industry standards such as WS-Reliability can be used to originate the request of a reliable invocation of a software service at the root level. In addition, as part of the present invention, an attribute can be associated to the interface of a software service to declare that the invocation of the service must be guaranteed. [See FIGURE 1 to see a depiction of an example software service interface]. Other attributes can be associated to a software interface declared to have guaranteed invocation, declaring the number of retries, the time between retries and other attributes determining the behavior of the automating platform. [FIGURE 4 shows an example of attributes that can be associated with a software interface to declare guaranteed invocation behavior.] Furthermore, these attributes can be overwritten for a contained software service, in the context of a containing composite service. This allows context-sensitive definition of guaranteed invocation behavior for services embedded within composite software definitions.

[Para 21] To support the automatic implementation of guaranteed invocation behavior for atomic software services, a wrapper interface must be implemented for all service libraries wanting to participate in guaranteed invocation of the associated atomic software service. The implementation of these libraries can simply use standard protocols, such as WS-Reliability, to delegate the guaranteed invocation of the associated service to an external system, beyond the address-space of the automation platform, or it can use proprietary messaging systems and protocols, to guarantee the at least once or the exactly once invocation of the associated atomic service. Here, following the known current art of messaging protocols, the automation platform

guarantees that if the same service request is passed to a library more than once, it is always passed with the same unique id.

[Para 22] To support the automatic implementation of guaranteed invocation for nested composite services, a persistent, nested context mechanism is devised. This context mechanism uses a set of service interfaces to provide persistence for the state of the composite service context corresponding to the invoked contained services and the overall state of invocation of the composite service, containing other services, nested to any level of depth. The use of a set of service interfaces for persisting the complete state of the invocation increases the flexibility and customizability of the automating system by providing a layer of encapsulation that allows the automating system to use different storage mediums such as direct file system vs. relational database.

[Para 23] To create a context for a composite service, first a directed graph, hereon referred to as the execution (or invocation) graph, is created that captures the dependencies of all the contained service nodes, based on the connectivity and data dependency of the services. Then, an object, hereon referred to as the Invocation Map is created, based on the execution graph, to hold the context of a composite service upon invocation. Additionally, modifications to shared data structures accessed within the definition of the corresponding composite service are stored in the Invocation Map as an object holding the context of the shared memory.

[Para 24] Upon the invocation of a composite service, the automation platform, instantiates an Invocation Map, corresponding to the execution graph created based on the dependency of contained services. Then, it determines the set of services that can be invoked next by traversing the execution graph in the order of node dependencies, with the nodes with no dependencies or dependencies to the input data of the composite service only, as the first set of nodes; and, the nodes with dependencies to prior nodes and the inputs of the composite service, as the second set of nodes, and so on. After preparing the input data for those services, the automation platform, stores the corresponding datasets within the Invocation Map, while the Invocation Map ensures the persistence of its complete state that at each step

corresponds to the state of invocation for the composite service. Then, the automation platform invokes the prepared services, while logging such attempt through the Invocation Map, and associating a unique key, that is also logged, to the invocation of each service, for the purpose of recovery from an unknown state, or in case one of the services fails due to a system error such as connection failure, or external system downtime. Hereon, we will refer to the invocation of each set of prepared services as a Hypercycle. Also, the Invocation Map holds the unique context associated to each atomic library, keyed by the identifier of that library, as part of its overall context.

[Para 25] If a composite service, within another composite service under invocation is encountered, the automation platform, creates an Invocation Map corresponding to the inner-composite service instance, and adds this map to a stack of Invocation Maps that is stored as part of the Invocation Map corresponding to the containing (i.e. the outer) composite service. Now, if another composite service is encountered within the above inner-service, the Invocation Map corresponding to that service will be added to the above inner-service, and so on to any level of depth of nesting.

[Para 26] When the invocation of a contained service within a composite service, regardless of the depth of nesting, fails due to some system failure, such as inability to connect to an external system due to temporary network failure, the automation platform places the entire stacks of Invocation Map directly, or indirectly contained within the root Invocation Map, that is the composite service instance with no parent composite service instance, on a persistent queue. A separate and synchronized thread of the automation platform, examines the queue, at configurable periods, determines the last set of invoked services that failed to complete, and for each of them considering the parameters declared within their interface definition, or the overwriting parameters within the containing composite node, attempts to re-invoke the service with the same unique id used for the invocation of the service prior to this attempt.

[Para 27] At this point, if the service, whose re-invocation is being attempted by the automation platform, is an atomic service, the library guarantees the

only once invocation of the that service as described earlier. However, if the service is a composite service, its Invocation Map is restored, and the automation platform, similarly continues the invocation of that composite service. If the invocation is completed, the automating platform returns to the current context, that of the outer composite service, and in the same manner continues the invocation of the outer composite service to the next set of services based on the execution graph. But, if the inner service's invocation fails due to some system failure, the automation platform, increments the number of unsuccessful invocation retires so far, return to the outer service's context, increments the number of unsuccessful retries so far, and so on all the way to the root composite service.